AD-A100 145 NORTHERN ILLINOIS UNIV DE KALB DEPT OF MATHEMATICAL --ETC F/G 9/2
IMPLEMENTING THE CONTINUED FRACTION ALGORITHM ON THE ILLIAC IV.(U)
1980 M WUNDERLICH F49620-79-C-0199
UNCLASSIFIED AFOSR-TR-80-1084 NL

1 OF 1
AD A
100145

END
DATE
FILMED
7-81
DTIC

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 80 1084	2. GOVT ACCESSION NO. AD-A100 145	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) IMPLEMENTING THE CONTINUED FRACTION ALGORITHM ON THE ILLIAC IV		5. TYPE OF REPORT & PERIOD COVERED FINAL
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Marvin Wunderlich, Mathematical Sciences Dept, Northern ILlinois University		8. CONTRACT OR GRANT NUMBER(s) F49620-79-C-0199
9. PERFORMING ORGANIZATION NAME AND ADDRESS Northern Illinois University, Mathematical Sciences Dept., DeKalb, IL 60115		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A6
11. CONTROLLING OFFICE NAME AND ADDRESS AFOSR/NM, Bolling AFB DC 20332		12. REPORT DATE 1980
		13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

AD A100145

LEVEL II

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

DTIC ELECTE JUN 12 1981 S C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The factoring of large composite integers has an important inverse relationship with the security of certain types of encrytion systems. If a particular code is based on a 100 digit composite number, the code can be considered secure for the length of time it would take to factor a 100 digit number on the fastest computer available using the best known factoring algorithm.
The continued fraction algorithm has generally been regarded as the best proven method known for factoring large integers. The research (continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

412386

20. Continued

performed by the principle investigator and described in this report was to produce a detailed running time analysis of the algorithm and obtain a model from which CPU time estimates could be obtained for the factoring of very large numbers. Since the fastest machines in use today are array processors, such as the ILLIAC IV and the English ICL-DAP, a feasibility study was conducted showing that the continued fraction method can be efficiently implemented on such a machine.

[redacted]

R

AFOSR-TR-80-1084

Final Technical Report for [redacted]

F49620-79-C-0199

IMPLEMENTING THE CONTINUED FRACTION ALGORITHM ON THE ILLIAC IV

Principal Investigator: Marvin Wunderlich

11 1980

I. Abstract

The factoring of large composite integers has an important inverse relationship with the security of certain types of encryption systems. If a particular code is based on a 100 digit composite number, the code can be considered secure for the length of time it would take to factor a 100 digit number on the fastest computer available using the best known factoring algorithm.

The continued fraction algorithm has generally been regarded as the best proven method known for factoring large integers. The research performed by the principal investigator and described in this report was to produce a detailed running time analysis of the algorithm and obtain a model from which CPU time estimates could be obtained for the factoring of very large numbers. Since the fastest machines in use today are array processors, such as the ILLIAC IV and the English ICL-DAP, a feasibility study was conducted showing that the continued fraction method can be efficiently implemented on such a machine.

II. The Research Objectives

2.1. To provide an empirical and, to what e tent possible, a theoretical analysis of the continued fraction algorithm for factoring compositite integers. The empirical analysis will be based on a statistical study of 2800 factorizations performed by the principal investigator over the past three years using an implementation of the continued fraction algorithm on the IBM 360, model 67, at Northern Illinois University. This analysis should, if possible, investigate any possible improvements to the algorithms.

2.2. To perform a feasibillty study of implementing the continued fraction algorithm on one of the high speed parallel computers in existence at this time. This study should, if possible, include the ILLIAC IV computer at Sunnyvale, California, the British DAP, recently installed at Queen Mary University in England, and the Cray I pipeline computer. Since not all algorithms can be effectively implemented on a highly parallel machine, it must be shown that the logic involved in the continued fraction algorithm would lend itself to a parallel implementation without significant loss of efficiency.

2.3. To project on the basis of the findings obtained in the pursuit of the previous two objectives the computer resources required both in computer time and high speed storage requirements, to factor numbers considerably larger than ever before attempted with this algorithm. These projections should determine how large a number should be essentially unfactorable using the continued fraction method.

2.4. To implement the continued fraction algorithm on one of the parallel machines analyzed in this project. This should only be done if the work can be carried out to it's conclusion during the time of the current research grant.

III. Status of the Research Effort

The results of the research effort have not at the time of this writing been adequately reported in scientific and technical publications. However, a paper is being prepared for eventual publication in a professional journal which adequately reports on the research accomplishments pertaining to the first three objectives and a draft of this paper is a part of this report. In January, 1980, it was decided not to implement a version of the continued fraction algorithm on the ILLIAC IV. The decision was made by the principal investigator after consulting with Daniel Slotnick of the University of Illinois, Glen Lewis of the Institute of Advanced Computation and Joe Bram of AFOSR. The following considerations were pertinent to that decision:

A. The continued fraction method is not the fastest theoretical method for factoring large numbers. Shroeppel's method should be thoroughly studied before a large amount of resources is committed to an implementation of the continued fraction algorithm.

B. Only one hour of computer time was budgeted for the implementation of this method on the ILLIAC IV and, although this may be enough time for the design and initial testing of the program, it would leave little time for any use of the program. There is little opportunity of obtaining free time on the ILLIAC IV and little promise of obtaining continued support for number factoring from granting agencies.

C. By the time the initial investigation was completed, there were only five months left to implement the program. It was felt that this was too short a time to promise a total working program. An extension of time was ruled out by the AFOSR.

The bulk of the research performed during the grant period consisted of obtaining a thorough analysis of the *continued fraction* algorithm. Consequently, the bulk of this report consists of a report on that analysis. What follows is a draft of a paper which will be submitted for publication containing the results of that analysis and projections for more extensive use of the algorithm. Following the draft is an appendix, not intended for publication, containing a similiar report on Schroeppel's new sieve method of factorization.

1. Introduction

It is generally believed that the continued fraction algorithm of John Brillhart and Michael Morrison [3] represents the most efficient proven method of factoring large integers. A new method of Richard Schroeppel promises to be faster than continued fractions for very large integers, but at the time of this writing, the author is not aware that any non-trivial numbers have actually been factored by Schroeppel's new method.

A description of Morrison and Brillhart's method can be found in [3,5]. Their program was originally written for the 360/91 at U.C.L.A. and a version of the program was implemented on the 360/65 at Northern Illinois University in 1975. Originally, the program was only capable of factoring numbers up to 30 decimal digits in length, but a number of improvements and modifications have been made by this author, and at the present time, we are able to routinely factor numbers up to 40 digits in length. The purpose of this paper is to describe this continued fraction implementation in detail and report on our factorization of 2800 integers, ranging from 13 to 42 decimal digits. The output from these runs have been saved and statistically analyzed and we used the analysis to predict the success of this method on even larger numbers.

2. Brief Description

The algorithm finds integers X and Y for which $X^2 \equiv Y^2$ (modulo N) where N is the number we wish to factor. If $N=pq$ where p and q are primes, then $pq \mid (X-Y)(X+Y)$ and each of the four cases

1. $p | X - Y$, $q | X + Y$
2. $p | X + Y$, $q | X - Y$
3. $pq | X - Y$
4. $pq | X + Y$

will occur with roughly equal probability and a factor may be discovered by computing $GCD(X - Y , N)$. To find the integers X and Y, we expand N in the simple continued fraction

$$N = < q_o, q_1, q_2, \ldots, q_{n-1}, \frac{\sqrt{N} + Pn}{Qn} >$$

and we compute the convergent A_n defined by

$$A_o = 1,\ A_1 = q_o,\ A_n = q_n A_{n-1} + A_{n-2} \pmod N,\ n > 1.$$

It can be easily shown that

(1) $$(-1)^n Q_n \equiv A_n^{\ 2} \pmod N$$

and

(2) $$0 < Q_n \leq 2 N.$$

Our next objective is to find a subset of the $\pm$ Q's--which we will rename $Q_1, Q_2, \ldots, Q_t$---whose product is a square. We then have

(3) $$X^2 = \prod_{i=1}^{t} \pm Q_i \equiv \prod_{i=1}^{t} A_i^{\ 2} = Y^2 \pmod N$$

and a factor may be found by the manner described above. To find the subset, we factor a large number of Q over a fixed set of m primes $p_1, p_2, \ldots, p_m$. If $Q_j = (-1)^{\gamma_{o,j}} p_1^{\gamma_{1,j}} p_2^{\gamma_{2,j}} \ldots p_m^{\gamma_{m,j}}$ we form the matrix

(4) $$M = [\varepsilon_{i,j}]$$

where $\varepsilon_{i,j} \equiv \gamma_{i,j} \pmod 2$. It has n rows and m columns where n is the number of Q we have factored and m is the number of primes involved in the factorization.

m is fixed and a fixed proportion of the Q's will completely factor over a given set of primes. Thus, if we have a sufficient number of Q's, we will ultimately factor enough of them to produce a linear dependency among the rows of M, and such a dependency corresponds to a set of Q's whose product has a prime factorization in which all exponents are congruent to 0 mod 2, and thus is a square. The existence of such a dependency can be guaranteed by generating enough factored Q's to produce a matrix M having more rows then columns. Experience has shown that matrices which are square, that is m = n, have several dependences, and each of them has a good chance of producing a factorization. The dependencies can be produced by doing a standard Gaussian elimination on the matrix, performing the same operations on an appended identity matrix (a "history matrix") and the non-zero columns of the history matrix will indicate which Q's are to be multiplied to produce the required square. For a very readable account of this procedure and an illustrative example, see Morrison and Brillhart [3].

Before proceding with a detailed algorithm and analysis, a few remarks are necessary.

Remark 1. It can be shown that whenever $p|Q_i$ for a prime p and for any i, then the Legendre symbol

(5) $\left(\frac{N}{p}\right) = 0$ or 1.

If it is 0, then N is factored. Thus, we need only divide the Q's by the p's for which $\left(\frac{D}{p}\right) = 1$ which is about one half of the primes.

Remark 2. The factoring strategy is actually more complicated than is indicated above. We choose two program parameters x , y

which satisfy $y \leq x^2$. We attempted to factor each Q by dividing out all primes p which satisfy (5) and satisfy $p \leq x$. Let this collection of primes be p_1 , p_2 , ..., p_m and call it the factor base. If, after dividing, we have

$$(6) \qquad Q = p_1^{\gamma_1} p_2^{\gamma_2} \cdots p_m^{\gamma_m} \bar{Q}$$

we have a complete factorization of Q whenever $\bar{Q} < y$. The choice of x and y will be discussed in the analysis later in the paper. If $\bar{Q} > x$, we discard that Q, because either Q is composite or is so large that it's contribution to a linear dependency is very improbable.

Remark 3. With this factoring strategy, it is no longer true that the number of columns in the matrix M, is fixed. At the start of the program, it's value is m, the number of primes in the factor base, but each time $p_m < \bar{Q} < y$ in (6) the prime $\bar{Q}$ is added as a column in the matrix M. In actual practice, the factored Q's are saved on a temporary file along with their corresponding value of A and the actual matrix is not formed in the computer memory until it is reasonably certain that a dependency will occur.

The Algorithm In Detail

The author will assume that the reader is familiar with the material contained in [3]. Since this program is an adaptation of Brillhart and Morrison's program, reference will be made to their paper frequently in the algorithm. An excellent treatment of this procedure can also be found in Knuth [1].

The following notations are used:

N	- the number being factored
FB = n	- the size of the factor base
P = x	- the largest prime in the factor base
UB	- the upper bound on the largest prime factor of a Q accepted
I	- counts the number of Q's which have been factored
J	- n plus the number of factorizations which involve primes greater than x
LEVEL = I/J	- estimates the "squareness" of the matrix M
T	- an input parameter which determines when LEVEL is sufficiently large to guarantee dependencies in the matrix M

The algorithm will not specifically say how to compute the Q's and A's. This can be obtained by referring to [1] or [3].

1. [Initialiization]. Read in N, the number to factor and the input parameters FB, UB, and T. Compute the <u>factor</u> <u>base</u> p_1 , p_2 ..., p_n which are the first n primes which satisfy (5). Set $I \leftarrow 0$, $J \leftarrow FB$ and rewind the file FACTS.

2. Generate the next Q and it's corresponding value A. Divide out from Q all the primes in the factor base, producing

(7) $$Q = (-1)^{\gamma_0} p_1^{\gamma_1} p_2^{\gamma_2} \cdots p_n^{\gamma_n} \overline{Q}$$

3. If $\overline{Q} \leq UB$, set $I \leftarrow I + 1$. Otherwise, go to step 2.

4. Write on the file FACTS, the numbers Q, A, γ_o, the primes p_i in (7) for which γ_i is odd and the co-factor $\overline{Q}$. If $\overline{Q} > P$, set $J \leftarrow J + 1$.

5. If LEVEL = I/J < T , go to step 1.

6. (Scan Program). Sort the file FACTS in ascending order on $\overline{Q}$. Let $\overline{Q}_i$ be the value of $\overline{Q}$ stored in the i-th record after the sort. Eliminate from the file those records for which $Q_i \neq 1$, $Q_i \neq Q_{i-1}$ and $Q_i \neq Q_{i+1}$. After the elimination, let

   NF = number of factorizations (records) left on the file

   NFL = number of factorizations left with $\overline{Q} > 1$

   NP = number of distinct primes > x involved in the factorizations

7. Read in the reduced file, form the NF x NP + n matrix M as described in (4). Row reduce M using the procedure outlined in [3] page 188 possibly producing one or more $A - \hat{Q}$ pairs where $A^2 \equiv \hat{Q}^2 \pmod{D}$. For each pair, compute F = GCD $(A - \hat{Q}, D)$. If $1 < F < D$, return F as a factor of D and STOP. Otherwise, set $T \leftarrow T + .02$ and go to step 2.

The principle difference between this algorithm and the one described in [3] is the introduction of a scanning procedure (step 6) between the collection of the factored Q's and the row reduction. This was suggested by Morrison and Brillhart [3 , pp 197 , 198] as a way to reduce the large amount of core required to

row-reduce the matrix. As they suggested, it also allowed for a much larger value for UB in order to take full advantage of possible matches, and the larger value of UB dictated a longer value of T. Surprisingly, this also permitted the use of a much smaller factor base, thereby reducing substantially the amount of computer time required to factor a given size number. Our choice of parameters were determined experimentally and showed that with the scan program implemented, the program was very insensitive to the choice of FB. For numbers of 23 digits or more, we used FB = 150 and UB - 1,000,000. For numbers less than 23 digits, we used FB = 75 or 100 and UB = 1,600,000 - 2,500,000. For all numbers factored, we found the value .95 to be a workable value for the parameter T. Without using the scan, Morrison and Brillhart recommended using a factor base as large as 650 and a value UB = 53000 for numbers of 40 digits. (See [3], table 2).

Numerical Results

In this section, we give a summary of the results of our 2797 factorizations. This summary is contained in Table 1 and presented by digit size and parameter value FB. The factorizations were performed over a long period of time. Our choice of parameters were not always consistent with the recommended values given in the last section. The following defines each column in Table 1.

DIGS - the number of digits in the numbers factored in this category

FB - the number of primes in the factor base

OBS - the total number of factorizations in the category.

LPF - the mean value of the largest prime number in the factor base.

UB - the median value of the upper bound.

NQ - the mean value of the total number of Q for which a factorization was attempted.

W - the mean value of the total work involved. ($W = 10^{-6} * NQ * FB$ - See explanation below)

NF - the mean value of the number of factored Q's produced.

FR - the mean of NF/NQ. This measures the fraction of the Q's which factored.

NP - the mean of the largest value attained by J in step 3 of the algorithm. It is the total number of distinct primes involved in the factorizations, assuming that all primes in the factor base occur, and not counting matches outside the factor base.

LV - the mean of NF/NP. This should generally be equal to T = .95 , unless the number was small. This will be discussed later.

SNF - the mean of the number of factorizations remaining after the scan routine was executed.

SNP - the mean of the number of primes involved after the scan. Matches are now accounted for.

SLV - the mean of SNP/SNQ.

TABLE 1

DIGS	FB	OBS	LPF	UB	NQ	W	NF	FR	N^P	LV	SNF	SNP	SLV
13	75	12	824	227500	328	.025	247	0.764	255	0.967	85.8	84.0	1.020
13	100	7	1186	434285	393	.039	312	0.810	318	0.980	115.3	110.1	1.042
13	150	1	1901	998001	329	.049	311	0.945	327	0.951	146.0	156.0	0.935
14	75	40	804	234250	417	.031	280	0.686	291	0.963	91.6	86.1	1.064
14	100	16	1165	425000	502	.050	363	0.732	374	0.971	125.1	116.6	1.068
14	150	2	1870	998001	583	.087	466	0.804	490	0.951	163.0	167.5	0.973
15	75	73	825	230273	594	.044	331	0.571	344	0.964	97.7	91.5	1.063
15	100	33	1168	391515	679	.068	423	0.635	431	0.981	131.6	118.4	1.107
15	150	2	2046	998001	627	.094	460	0.735	484	0.950	218.0	154.5	1.425
16	75	85	808	237294	776	.058	364	0.485	379	0.961	101.1	94.2	1.069
16	100	33	1141	383636	897	.090	481	0.551	489	0.984	141.5	123.2	1.140
16	150	5	1999	998001	904	.136	590	0.662	618	0.955	166.4	170.6	0.975
17	75	118	813	234746	1032	.077	403	0.406	419	0.960	106.5	97.4	1.089
17	100	34	1172	391765	1232	.123	544	0.456	553	0.984	152.1	128.6	1.177
17	150	5	1833	998001	1207	.181	634	0.527	663	0.959	156.2	160.8	0.982
18	75	138	800	231086	1428	.107	437	0.321	456	0.960	111.4	100.5	1.103
18	100	68	1171	381636	1696	.170	611	0.375	620	0.984	161.3	132.8	1.206
18	150	2	1925	998001	1376	.206	710	0.517	747	0.950	172.5	178.5	0.966
19	75	136	807	232132	2030	.152	486	0.252	506	0.959	117.0	104.1	1.119
19	100	68	1168	380294	2453	.245	706	0.308	715	0.987	185.9	143.7	1.283
19	150	7	1902	998001	2014	.302	785	0.397	824	0.953	180.4	183.0	0.984
20	75	157	812	236815	2775	.208	523	0.198	545	0.960	123.9	107.8	1.145
20	100	69	1182	380290	3463	.346	784	0.239	795	0.985	201.9	152.1	1.316
20	150	4	1911	998001	2554	.383	823	0.333	868	0.946	177.8	184.5	0.963
21	75	156	805	236859	4165	.312	571	0.147	596	0.959	130.2	112.0	1.157

TABLE 1 (Con't)

DIGS	FB	OBS	LPF	UB	NQ	W	NF	FR	NP	LV	SNF	SNP	SLP
21	100	75	1155	378400	4477	.448	821	0.195	837	0.981	203.4	154.4	1.308
21	150	6	1874	998001	4035	.605	831	0.233	875	0.951	177.8	177.2	1.016
22	75	148	805	234189	5229	.394	580	0.118	605	0.957	129.1	112.0	1.148
22	100	70	1152	377857	5915	.519	819	0.145	842	0.972	191.7	152.7	1.245
22	150	3	2072	998001	4472	.670	955	0.216	1007	0.948	189.7	194.0	0.975
23	100	4	1213	430000	4939	.494	760	0.162	784	0.969	166.0	143.0	1.152
23	150	191	1902	998001	6218	.933	999	0.170	1054	0.948	193.4	196.3	0.983
24	75	1	727	360000	7308	.549	564	0.077	600	0.940	98.0	103.0	0.951
24	100	7	1196	440000	13228	1.32	944	0.076	980	0.961	196.9	161.9	1.204
24	150	231	1909	998001	8539	1.28	1063	0.132	1120	0.948	198.9	200.2	0.991
25	75	1	857	360000	17559	1.32	715	0.041	752	0.951	119.0	113.0	1.053
25	85	1	857	360000	17407	1.48	746	0.043	785	0.950	111.0	116.0	0.957
25	100	6	1203	500000	18547	1.85	883	0.058	925	0.953	171.2	153.0	1.106
25	150	166	1904	998001	12382	1.86	1138	0.098	1200	0.948	205.1	205.2	.998
26	75	2	878	305000	13488	1.01	909	0.073	956	0.950	162.0	158.5	1.027
26	100	3	1125	640000	14595	1.46	842	0.058	890	0.946	138.3	139.7	0.990
26	150	151	1902	998001	15678	2.35	1183	0.080	1248	0.948	210.2	208.4	1.007
27	100	1	1093	360000	38358	3.84	785	0.020	842	0.932	155.0	154	1.006
27	150	107	1893	998001	23307	3.50	1258	0.060	1325	0.949	220.0	214.3	1.024
28	100	3	1137	453333	37068	3.71	916	0.025	962	0.951	161.0	149.3	1.078
28	150	83	1918	998001	34184	5.13	1335	0.042	1408	0.948	228.3	220.7	1.033
29	100	1	1237	360000	60990	6.10	893	0.015	948	0.942	174	159	1.094
29	150	73	1922	998001	44312	6.65	1378	.0345	1452	0.950	230.9	220.6	1.036
30	100	2	1053	360000	48392	4.84	838	.01740	882	0.950	152	145.5	1.043
30	150	48	1863	998001	65446	9.82	1439	.0241	1517	0.949	240	229	1.047

TABLE 1 (Con't)

DIGS	FB	OBS	LPF	UB	NQ	W	NF	FR	NP	LV	SNF	SNP	SLP
31	150	33	1886	998001	87508	13.13	1497	.0184	1573	0.952	246	230	1.066
32	150	21	1900	998001	137072	20.56	1528	.0122	1608	0.950	247	234	1.052
33	150	15	1893	998001	225301	33.80	1597	.0081	1681	0.950	244	234	1.041
34	150	16	1932	998001	315132	47.27	1661	.0062	1746	0.950	235	231	1.072
35	150	5	1920	998001	370431	55.56	1540	.0042	1631	0.944	236	235	1.007
36	150	2	1764	998001	371398	55.7	1595	.00446	1682	0.948	238	229	1.037
37	150	2	1771	998001	745430	111.8	1619	.00233	1722	0.941	246	239	1.025
38	150	4	1888	998001	1459781	219.0	1691	.00157	1785	0.948	242	238	1.016
39	150	10	1948	998001	1609369	241.4	1743	.00112	1841	0.947	241	238	1.008
40	150	17	1905	998001	2665175	399.8	1765	.00086	1869	0.944	248	246	1.008
40	200	2	2514	998001	1489049	297.8	2060	.00138	2184	0.944	341	323	1.056
41	150	17	1936	998001	3026875	454.0	1746	.00067	1848	0.945	247	243	1.012
42	150	7	1809	998001	4747294	712.1	1705	.00040	1807	0.943	243	241	1.005

Before continuing with the analysis, a few remarks should be presented about the raw data contained in Table 1. The program, which factored all these numbers, is a completely automatic, self-executing system, which was designed to use up all the idle time on the machine in the early morning hours and on weekends. The parameters were chosen to minimize CPU time for large numbers. The values of Q were generated and factored by a program called RESIDUE, which executed over and over in 20 minute runs until the number was factored. When the value of T exceeded .93, the scan and the row reduction was performed by a program called GAUSS between each execution of RESIDUE. When the number was factored, the output was placed on a disk file and a program was submitted, which read the file and continued the number theory project which needed the factorization. The table shows that for numbers of 25 digits or more, the number factored when the value of LV was very near .95 and the scan program produced a reduced matrix, which was very nearly square (SLV). In that case, the row reduction produced very few dependencies and, consequently, very little time was wasted. For smaller numbers, however, more attention was paid to making the system fail-safe and minimizing the number of individual jobs which had to be run. (This was partly to show mercy on the computer operators who periodically had to purge all of these jobs from the system.) For these numbers, a value of $T = .95$ was used with an increment of .04 or .02. This produced, in many cases, a scanned matrix with considerably more rows than columns. Often 30 or 40 dependencies were produced by

GAUSS. This "over kill" indicates that the efficiency of the system could be improved for small numbers of 20 digits and less.

The amount of work (W) performed was just 10^{-6} times the number of divisions required to factor all the Q's. This ignores the time required to scan and row reduce the matrix, which is insignificant compared to the factoring time. On the IBM 360/65, each division takes about 64 μ sec for all size numbers. Therefore, the 7th column of Table 1 is approximately the number of CPU minutes required to factor the number. Note that about 400 CPU hours were used by numbers 35 digits and larger.

<u>The Factoring Ratios</u>

The efficiency of this factoring method depends very strongly on the fraction of numbers completely factored by the algorithm. In this section, we will analyze this ratio using Dickman's function and show how closely the results compare with the actual factorizations.

The two parameters which govern the factoring strategy are P, the largest prime in the factor base, and UB, the upper bound. Since the typical Q is about N, we will let $\alpha = 2 \log P / \log N$ and $B = 2 \log UB / \log N$. Therefore, we have

$$Q^{\gamma} = P$$

and

$$Q^{\beta} = UB$$

for the typical Q. Let $r(\alpha)$ be the fraction of Q, which factor completely over the primes less than P and let $r(\alpha, \beta)$ be the fraction of Q for which the largest prime factor is less than UB and all other prime factors are less than P. $r(\alpha)$ should compare closely with Dickman's function $F_1(\alpha)$ which is the limiting

fraction of numbers $\leq N$ whose largest prime factor is less than N^{α}. (In our situation, our Q's consist of products of primes satisfying (5) which is about half of the primes. However, one can easily show that for each p in the factor base, the probability that $p|Q$ is about $\frac{2}{p}$ rather than $\frac{1}{p}$ and these two facts have cancelling effects.) It is a bit more complicated to estimate $r(\alpha,\beta)$. If $\beta = 2\alpha$, (that is $UB = P^2$) $r(\alpha,\beta)$ should be closely approximated by Knuth and Trab Pardo's function $G(\alpha)$ [2] which is the limiting ratio of number $< N$ having it's largest prime factor less than α^2 and second largest prime factor less than α.

Fortunately, our values of P and UB were chosen so that the relationship between α and β is closely linear. Figure I shows a scattesgram of α verses β. The ρ - Phearson correlation is .99506 (a sociologists dream), the y intercept is .00068 and the slope is 1.83271. Thus, our data consistently used values of α and β which satisfied $\beta \doteq 1.83271\alpha$ and our observed values of $r(\alpha,\beta)$ should be slightly less than $G(\alpha)$.

TABLE 2

1	2	3	4	5	6	7	8	9
OBS	Range of α	Mean α	Mean α	Ave # Digits	Mean $r(\alpha)$	Mean $r(\alpha,\beta)$	$F_1(\alpha)$	$G(\alpha)$
9	.15 - .1599	.15709	.28942	41.8	.0000127	.0004137	.00000629	.000402271
77	.16 - .1999	.17702	.32356	37.8	.0001073	.0027796	.00005424	.00256299
251	.20 - .2399	.22499	.41244	29.6	.0016855	.0295487	.00157812	.040715800
805	.24 - .2799	.26422	.48557	24.6	.0083623	.0994837	.00821723	.138541144
881	.28 - .3199	.295630	.542184	21.3	.01945861	.1847054	.02099718	.263330855
402	.32 - .3599	.338486	.621050	18.1	.04868968	.3398609	.053190253	.467794433
208	.36 - .3999	.37817	.69466	16.2	.08805736	.4941699	.098029591	.633893438
116	.40 - .4399	.41705	.76470	14.8	.13860717	.6259784	.15525344	.778718019
43	.44 - .4799	.45565	.832703	13.7	.19882978	.7390249	.219857085	.884806329
4	.48 - .5199	.49501	.900750	13.3	.20688597	.7876878	.296430227	.987394607

Table 2 demonstrates the close relationships between our observed values $r(\alpha)$ and $r(\alpha\,,\beta)$ and the functions $F_1(\alpha)$ and $G(\alpha)$. The data was divided into 10 subsets, depending on the value of α. The range of α is given in column 2 and within the range, the means of α , β , digit size, $r(\alpha)$ and $r(\alpha\,,\beta)$ is tabulated in columns 3 through 7. The values of $F_1(\alpha)$ and $G(\alpha)$ was computed from Table 1 in Knuth and Trab Pardo [2] page 340, where the mean of α given in column 3 were used as the argument. (Geometric interpolation was used.) Agreement was seen to be within a factor of 2.

We can use these factoring ratios to obtain a theoretical model for the algorithm, from which a running time estimate can be derived. If we know that whenever LEVEL, the ratio of the number of primes involved in the factorizations to the number of factorizations, exceeds the parameters T, the scanned and row reduced matrix will have zero rows, we can write

(8)
$$T = \frac{NF}{FB + NF\ (1 - \frac{r(\alpha)}{r(\alpha,\beta)})}$$

or

(9)
$$NF = \frac{T \cdot FB}{1 - T(1 - \frac{r(\alpha)}{r(\alpha,\beta)})}$$

$$\frac{1}{r(\alpha,\beta)} \cdot \frac{T \cdot FB}{1 - T(1 - \frac{r(\alpha)}{r(\alpha,\beta)})}$$

and

(10)
$$\frac{1}{r(\alpha,\beta)} \cdot \frac{T \cdot FB^2}{1 - T(1 - \frac{r(\alpha)}{r(\alpha,\beta)})}$$

where NQ and ND are the number of Q and the total number of divisions needed to factor the number. (8) is true since a column is added to the matrix whenever a Q factors, but not over primes contained in the factor base.

(10) can be written

$$ND = C \cdot FB^2$$

where

$$C = \frac{1}{r(\alpha,\beta)} \cdot \frac{T}{1 - T(1 - \frac{r(\alpha)}{r(\alpha,\beta)})}$$

is a constant depending only on the parameter values α, β and T. If we choose α, β, $r(\alpha)$ and $r(\beta)$ to be the values tabulated in line 1 of Table 2, and choose our matrix threshold value to be T = .95, we get C = 29008. If we use the prime number theorem to estimate

$$FB = \tfrac{1}{2}N^{\alpha/2}/\tfrac{\alpha}{2} \log N = N^{\alpha/2}/\alpha \log N$$

using the assumption that only half of the primes satisfy (5), we can write

$$\text{(11)} \qquad ND = \frac{C\ N^{\alpha}}{\alpha \log N} = C\ N^{x}$$

where $$x = \alpha - \frac{\log \alpha}{\log N} - \frac{\log \log N}{\log N}$$

$$\doteq 0.128079395$$

for $N = 10^{40}$. Such an estimate can be useful in predicting the amounts of computer time required to factor numbers larger than 42 digits. we will first look more closely at our assumptions about the use of the prime number theorem to estimate the size of the factor base and our use of the matrix threshold value T = .95.

Table 3 attempts to justify the use of the prime number theorem to estimate the size of the factor base as a function of α. There are

only 5 rows since only 5 values of FB were commonly used. Within each value of FB, the average value of the largest prime in the factor base, LPF, is given in column 3, the estimate LPF/log(LPF) is averaged in column 4, and the ratio FB/(LPF/log(LPF)) is averaged in column 5. If the estimate LPF/log(LPF) is valued, the ratio in column 5 should be .5, since roughly half of the prime satisfy (5). We see that for numbers of our size, .6 seems closer. (Something over .5 is expected, due to the error term of the prime number theorem.) We also see that the size of FB does not seriously alter the value of column 5. Thus, a better estimate for FB would be

(12) $\quad FB = 1.2\ N^{\alpha/2}/\alpha \log N$

TABLE 3

# Observations	FB	Average of LPF	Average of LPF/log(LPF)	Average of FB · log(LPF) / LPF
1067	75	808.6	120.7	0.626
5	85	881.0	129.9	0.657
487	100	1165.4	165.0	0.609
1236	150	1904.2	252.1	0.597
2	200	2514.0	321.1	0.623

It is more difficult to justify the use of the value T = .95. At the time of this writing, the author does not have a theoretical or even a heuristic argument for it's value. It was found over many years of experience that the matrix produced dependencies as soon as LEVEL = I/J approached the value .95. As Table 1 shows, the

average value of LV is very near .95 for numbers 25 digits or more, but larger values appear for smaller numbers. Table 4 shows some averages when the data was grouped by LV.

TABLE 4

Range of LV	Observations	Average number of Digits	Average SLV	Average # of Dependent Rows
.93 - .9399	120	23.4	0.933	9.4
.94 - .9499	335	28.5	1.002	14.5
.95 - .9599	1045	24.1	1.029	20.7
.96 - .9699	853	18.9	1.119	24.1
.97 - .9799	76	22.1	1.197	41.9
.98 - .9899	29	21.4	1.381	66.7
.99 - .9999	320	18.5	1.281	54.8
1.00	15	16.1	1.257	38.5

The large numbers in the sample are reflected in line two of the Table and the high values of LV predictably result in values of SLV, which are significantly greater than 1 and produce more than an adequate number of dependencies. In an effort to learn how many dependencies are necessary to factor a number, the data was grouped by the number of dependencies and Table 5 shows the result of this grouping. We see from columns 5 and 6 that, in general, large numbers of dependencies resulted from large values of LV and SLV and this occurred in smaller numbers where a large value of LV was forced in order to avoid failures. There were some notable examples, however,

TABLE 5

1	2	3	4	5	6
Range of # Dependencies	Average # Dependencies	Number of Factorizations	Average Number of Digits	Average LV	Average SLV
1 - 5	3.48	98	33.04	0.943	0.965
6 - 10	8.14	195	26.71	0.945	0.964
11 - 15	13.16	381	24.31	0.951	0.996
16 - 20	18.02	563	22.45	0.953	1.031
21 - 25	22.79	501	21.65	0.956	1.077
26 - 30	27.82	357	21.65	0.956	1.106
31 - 35	32.66	210	21.45	0.960	1.137
36 - 40	37.83	103	20.69	0.968	1.177
41 - 45	42.78	90	19.42	0.978	1.202
46 - 50	47.96	67	19.80	0.983	1.243
51 - 55	52.65	46	19.75	0.985	1.273
56 - 60	57.79	38	20.26	0.989	1.320
61 - 65	63.04	46	20.58	0.987	1.338
66 - 70	67.89	29	21.27	0.988	1.370
71 - 75	73.23	26	21.45	0.991	1.407
76 - 80	77.76	21	20.34	0.990	1.421
81 - 85	82.40	10	21.98	0.987	1.422
86 - 90	88.44	9	19.91	0.990	1.458
91 - 95	92.20	5	21.22	0.987	1.476
96 - 100	99.00	2	20.73	0.991	1.506

for which a very large number of dependencies were still not sufficient to factor a number. Table 6 shows ten numbers which failed to factor after computing more than 25 dependencies. The column labeled MM is the value of a multiplier, which was chosen using the method in [3, p 194]. DF is a number of dependencies which was not sufficient to factor MM · N and DS was a number of dependencies which was sufficient.

TABLE 6

N	MM	DF	DS	FB	UB	LV	SLV
70549 26288 08101	1	65	74	75	400	1.092	1.634
5815 07793 34883	1	43	50	75	400	0.992	1.308
6 45510 11269 54357	1	32	41	150	999	0.970	0.994
195 13504 96582 27529	1	31	44	75	400	1.010	1.362
39471 29381 36701	46	30	43	75	500	1.000	1.253
5494 11882 38017	3	30	41	75	500	1.000	1.253
10539 87857 96651	74	28	41	75	500	1.041	1.333
58 92917 77150 41989	5	26	43	75	500	1.000	1.287
1 36393 73952 62593	1	26	32	75	400	0.991	1.163
24932 76465 39031	15	26	35	75	500	1.000	1.217

Projections

We will now use the analysis in the previous section to predict how much computer time will be required to facto› › number of a given size using the continued fraction method. For this projection, we

used $\alpha = .15709$, $\beta = .28942$, $r(\alpha) = .0000127$ and $r(\alpha,\beta) = .0004137$ taken from line 1 of Table 2. We are basing the projection on only 9 observations, but it is felt that the smallest possible value of α should be used. Using (9) and (10), these values yield

$$NF = 12 \cdot FB$$

$$NQ = 29008\ FB$$

and

$$ND = 29008\ FB^2.$$

For FB, we use (12). Table 7 illustrates our projections.

TABLE 7

Number of Decimal Digits	FB	NF	NQ	ND	Time
35	53	640	1,540,000	82,000,000	24 sec.
40	114	1,400	3,300,000	380,000,000	115 sec.
50	561	6,700	16,000,000	$9. \times 10^{9}$	45 minutes
60	2853	34,000	83,000,000	2.36×10^{11}	19 hours
70	14923	178,000	430,000,000	6.46×10^{12}	22 days
78	56916	680,000	1,650,000,000	9.400×10^{13}	10 months

The last row (78 digits) were chosen to predict the time needed to factor $F_8 = 2^{256} + 1$, a 78 digit number known to be composite for which no known factor exists.* The last column predicts the CPU time consumed if each division of a p by a Q takes 300 mano-seconds. This extraordinarily low estimate assumes that the division process can be carried out on a very fast array processor such as the ILLIAC IV or the English ICL - DAP. More will be said about this later.

* RICHARD BRENT has just shown that $F_8 = 1238926361552 \times P$ where P is a 62 digit prime number.

It is clear from Table 2 that efficiency is improved for larger numbers by choosing smaller values of α. It is also clear that the best factoring strategy is obtained by selecting $\beta = 2^{\alpha}$. Otherwise, Q's are factored completely and then rejected. In our factorizations, β was chosen smaller to reduce the amount of external storage needed to save the factored Q's. Table 8 demonstrates how the optimum value of α can be determined for a 40 digit factorization assuming $\beta = 2^{\alpha}$.

TABLE 8

α	$r(\alpha)$	$r(\alpha, 2\alpha)$	LPF	FB	NF	NQ	ND
1/5	.0003547297	.01241348	10,000	651	8,021	646,229	420,980,792
1/5.25	.000172091	.006760867	6,449	441	5,649	835,633	368,641,497
1/5.5	.000083488	.0036822	4,328	310	4,119	1,118,667	347,002,583
1/5.75	.000040503	.002005479	3,007	225	3,094	1,542,834	347,659,051
1/6	.000019650	.0010923	2,154	168	2,384	2,183,300	367,708,466
1/7	.0000008746	.00007139	719	66	1,011	14,170,359	930,095,061

Using this method for optimizing α for various size numbers produced projections which didn't differ dramatically from those reported in Table 5 and are summarized in Table 9.

TABLE 9

N	α	LPF	FB	NF	NQ	ND
10^{40}	1/5.5	4,330	310	4,119	1,120,000	347,000,000
10^{50}	1/6	14,700	920	13,000	11,900,000	1.09×10^{10}
10^{60}	1/6.75	27,800	1,630	24,700	175,000,000	2.85×10^{11}
10^{70}	1/7	100,000	5,210	80,300	1,113,000,000	5.86×10^{12}
F_8	1/7.25	206,000	10,100	158,000	4,660,000,000	4.77×10^{13}

Parallel Machines

The fastest computers available today make use of a high degree of parallelism. In such a machine, hundreds or even thousands of individual processing units are capable of executing a single instruction stream on independant data sets. Two such machines will be briefly described in this section.

The ILLIAC IV has 64 parallel arithmetic processing elements, (PE's), each of which is roughly comparable in function to the arithmetic unit of a conventional computer. The PE's are synchronized and all perform the same instruction simultaneously (in "lock step"), but with different data. Each PE is capable of fetching and storing data in it's own processor memory consisting of 2048 64-bit storage registers. Under program control, any of the processor memories can be blocked from excuting any given set of instructions; thereby providing flexible program control. However, when a large number of processors are blocked from doing useful work, the efficiency of the execution is reduced. One still pays for executing all 64 processors. When all processors are operating at 100% efficiency, the ILLIAC IV was found to be 400 times as fast as the IBM 360, model 67. Since for 40 digit numbers, we found that each division in step 2 of the continued fraction algorithm takes about 64 μsec. on the IBM 360, model 67, we can expect each division on the ILLIAC IV to take about 64/400 μsec. or 160 mano-seconds. One would have to double that figure for factoring numbers longer than 40 digits, since 64 bit numbers can only hold numbers up to 19 decimal digits.

The ICL DAP is a fast array processor which has a configuration similiar to that of the ILLIAC IV. There are two machines presently

in existence, one in Stavenage, England with 1024 parallel processors, and one at Queen Mary University (a branch of the University of London) with 4096 parallel processors. Each processor has it's own memory consisting of 1024 bits. Each processor is capable of simultaneously executing a single instruction set on all or a pre-determined subset of the 4096 processors. Again, the selection of which of the processors are active can be done under program control. Although the instruction set of the DAP consists of very primitive bit minipulation instructions, there exists fast optimally coded soft-ware for doing integer arithmetic in the processors. In a private communication with the author, Dr. S.F. Reddaway of ICL claimed that the time, in mano-seconds, to divide a number of P bits by a divisor of Q bits producing a quotient and remainder is

$$\text{Time} = P \cdot Q \cdot 1\tfrac{1}{2} \cdot 200 \tag{13}$$

and this produces 4096 simultaneous results. To factor a 60 digit number, we would required $P = 100$ and $Q = 15$. Using (13), we see that assuming 100% usage of all the processors, we would require 110 mano-seconds for each division. This justifies the time estimates in Table 5, which were based on a divide time of 300 mano-seconds.

We must now show that the continued fraction algorithm can be implemented on a parallel processor with a high level of efficiency. We know that not all algorithms can be so implemented.

For example, the Pollard Monte Carlo method [4] is inherently sequential. Each term of the sequence depends on the previous term, and although two sequences can be simultaneously generated, there is no effective way to make full use of 4096 processors. In the continued fraction algorithm, however, the generation of the Q's which must be executed sequentially, takes a small fraction of the total execution time. Most of the time is spent factoring the Q's and this could be done in parallel processors.

The following is an algorithm for performing the factoring in the continued fraction algorithm on a computer having M parallel processors. We assume that each processor memory contains registors Q, QUOT, REM and POWER. Q is large enough to hold a generated value of Q_i and QUOT is large enough to hold the largest prime in the factor base. We also assume that each processor memory contains registers sufficient to hold a copy of the primes in the factor base $p_1, p_2, \ldots, p_F$. This is unreasonable for the small processor memories of the DAP, but the algorithm can be suitably modified for an individual machine. (The primes can be processed in batches, for example.) We merely wish to give a general algorithm in order to discuss processor efficiency. We also assume that each processor has a flag P-FLAG, which disables execution of that processor when P-FLAG = OFF.

ALGORITHM P1

Step 1. Compute the primes in the factor base, $p_1, p_2, \ldots, p_F$

and place them in each of the processor memories.

Step 2. Compute M values of Q using the continued fraction expansion of N and place one in each processor memory. Set all P-FLAG's to ON. Set $i \leftarrow 1$.

Step 3. Perform 3.1 through 3.5 for $i = 1, 2, \ldots, F$.

3.1 Set POWER $\leftarrow$ 0.

3.2 Divide Q by p_i, place the quotient in QUOT and remainder in REM.

3.3 For those processors for which REM = 0

Set POWER $\leftarrow$ POWER + 1

Set Q $\leftarrow$ QUOT

Otherwise

Set P-FLAG $\leftarrow$ OFF.

3.4 If any P-FLAG is ON, go back to step 3.2.

3.5 Place $p_i \leftarrow$ POWER.

Step 4. For each processor satisfying $Q < p_F^2$, store the value Q and those primes for which $p_i > 0$. These are the Q which factored completely.

Remark: Step 3.5 replaces the values p_i with the power e for which $p^e \| Q$. It is assumed that a copy of the primes are retained so the primes themselves can be stored in step 4. Q is also replaced by QUOT so that it's original value must also be retained.

In step 3.2, M divisions are performed, but for small primes, many of those divisions are executed when most processors are off. Thus, a great deal of inefficiency is apparently tolerated. To quantify the efficiency of this algorithm, we will let D_M be the total number of divisions executed by step 3.2 of the algorithm.

For each of these divisions, let r be the fraction of the processors which are currently enabled. Then let D_R be the sum of these ratios. The efficiency of the program is clearly D_M / D_R. The efficiency for a single prime p can be estimated by the formula

$$(14) \qquad \frac{D_M}{D_R} = \frac{1 + \frac{1}{p} + \frac{1}{p^2} + \frac{1}{p^3} + \ldots + \frac{1}{p^t} + \frac{1}{M}}{t + 1 + \frac{M}{p^{t+1}}}$$

where $t = [\log M/\log p]$. This uses the fact that the expected fraction of numbers divisible by p^α is $1/p^\alpha$ and that if $p^\beta > M$, the probability that one among M numbers is divisible by p^β is M/p^β. Small primes are the most inefficient. If $p = 2$ and $M = 4096$, (14) yields an efficiency ratio of 0.148, where as when $p = 20011$ the efficiency is 0.830. Of course most primes in the factor base are large and the sum of the efficiency ratios will reflect the higher values rather than the lower ones. The total efficiency E over all primes in the factor base can be estimated by the formula

$$(15) \qquad E = \frac{D_M}{D_R} = \frac{\sum_p \left(1 - \frac{1}{p^{t+1}}\right) / \left(1 - \frac{1}{p}\right)}{\sum_p \left(1 + t + \frac{M}{p^{t+1}}\right)}$$

where $t = t_{M,p} = [\log M/\log p]$ and the sums are taken over all primes in the factor base.

We show the estimated overall efficiency for a simulated 40 digit and 60 digit factorization in Table 10. F is the number of primes in the factor base taken from Table 7 and D_M, D_R and E are

from (15). The primes in the factor base were computed by letting $p_1 = 2$ and $p_k = p_{k-1} \cdot 2 \cdot \log p_{k-1}$. This produced prime-like numbers very close in density to the factor bases described in Table 3. As expected, a large set of processors and small factor bases produce the most inefficient factoring strategy.

TABLE 10

	40 DIGITS				60 DIGITS			
# Processors	F	D_M	D_R	E	F	D_M	D_R	E
64	114	153.84	117.55	0.764	2853	2906.2	2899.4	0.998
1024	114	251.82	116.31	0.462	2853	3204.6	2858.0	0.892
4096	114	267.30	116.31	0.435	2853	3741.6	2856.0	0.763
*16374	114	286.00	116.31	0.407	2853	4847.1	2855.6	0.589

* A DAP-like machine is being designed for NASA having 16374 parallel processors.

One can avoid processor inefficiency if one is willing to employ large amounts of temporary storage. The time consuming aspect of the continued fraction algorithm is the trial division, and one can trial divide in parallel processors with 100% efficiency as long as complete factorizations are not required. A sketch of such a factoring is described in Algorithm P2, in which the work is performed on two processors; one expensive parallel machine with M processors and one inexpensive sequential processor.

ALGORITHM P2

A. Compute all the Q's needed to factor N along with their associated values of A and put them on an external storage device. For this, the sequential machine is used.

B. On the parallel machine, read in the Q's in M-sized batches and divide each Q by each p in the factor base exactly once. For each Q, store on external storage the record

$$(Q, A, p_1, p_2, \ldots, p_k)$$

where each of the p's divide Q.

C. On the sequential machine, read in each Q and divide it by the p's as often as necessary in order to attempt the complete factorization of Q.

It is clear that B can be done on the parallel processor with 100% efficiency. Step C requires only about $2 \log \log Q$ divisions for each Q. For 60 digit numbers, $\log \log Q$ never exceeds 5, so that from Table 5 only 830,000,000 divisions are performed. This is far less than the 23.6×10^{10} trial divisions needed. The parallel processor could also be employed for step C to make that calculation even less time consuming.

As a result of these projections, we can conclude that an implementation of the continued fraction algorithm on a highly parallel machine, such as the DAP, could concievably give us the capability of factoring numbers of 55 decimal digits, whereas with our present non-parallel machines, we find that 43 or 44 digits is a practical upper limit, without using unreasonable amounts of CPU time.

REFERENCES

1. D.E. Knuth, The Art of Computer Programming, v.2, Seminumerical Algorithms, second edition, Addison-Wesley, in press.

2. D.E. Knuth and Trabb Pardo, "Analysis of a Simple Factorization Algorithm", Theoritical Computer Science 3 (1976) pp. 321-348.

3. M.A. Morrison and J. Brillhart, "A Method of Factoring and the Factorization of F_7,", Math. Comp., v. 29, 1975, pp. 183-205.

4. J.M. Pollard, A Monte Carlo Method for Factorization, Nordisk Tidskr Informationshehandling (BIT) 15 (1975), pp. 331-334.

5. M.C. Wunderlich, "A Running Time Analysis of Brillhart's Continued Fraction Algorithm," Number Theory, Carbondale, 1979, Lecture Notes in Mathematics, v. 751, ed. by M.B. Nathanson, 1979, pp. 328-342.

APPENDIX. Schroeppel's Method

The principal investigator was not aware of the details of this factoring method until the fall of 1979. In December of 1979, he discussed the method with R. Schroeppel in California, and the following is a brief report on the method. Since the procedure was not discovered or implemented by the principal investigator, this section of the report is not currently intended for publication and the enclosed analysis is intended for AFOSR personnel only.

Brief Description

It was mentioned before that the most time-consuming aspect of the continued fraction method is the trial division, and yet none of the information produced by those divisions is actually used--the quotient and remainder are both discarded. The program could be substantially improved if we could know in advance with Q's are divisable by what p's so we would not have to "trial" divide. Schroeppel's method essentially does this.

In this section, let D be the number to factor. Instead of generating Q's from the continued fraction expansion of $\sqrt{D}$, we let $K = [\sqrt{D}]$ and generate Q's defined by

(S1) $$Q = (K+A) + (K+B) - D$$

where

A ranges over the interval $\left(-\frac{S_1}{2}, +\frac{S_1}{2}\right)$

and

B ranges over the interval $\left(-\frac{S_2}{2}, +\frac{S_2}{2}\right)$

This allows us to compute $S_1 S_2$ values of Q where

(S2) $$Q < \frac{S_1 + S_2}{2}\sqrt{D} \quad \text{and} \quad S_1 > S_2.$$

Furthermore, if we fix B, the values of Q are an arithmetic expression in A and we can factor them using a sieve. This eliminates trial division. We must, however, find a subset of the Q's for which ΠQ_i as well as $\Pi(K+A)(K+B)$ is a square so that

(S3) $$X^2 = \Pi Q_i = \Pi[(K+A)(K+B)-D] \equiv (K+A)(K+B) = Y^2 (\text{mod } D).$$

To find the subset, we factor the Q's as before over a base of primes $p_1, p_2, \ldots, p_m$ and each factored Q produces one row in our 0-1 matrix M. The first $m=\Pi$ columns of the matrix represent the m primes in the factor base; the next N_2 columns represent the new primes added because of "type 2" factorizations that added a new prime $< p_m^2$. We also include S_1 columns, one for each possible value of A and B. (See illustration 1.)

<u>Illustration 1</u>

$$
\begin{array}{l}
\quad p_1\ p_2 \cdots p_m\ q_1\ q_2 \cdots q_{N_2}\ s_1\ s_2 \cdots s_{S_1} \\
Q_1 \quad 1\ \ 0\ \ \ldots 0\ \ 0\ \ 1\ \ \ldots 0\ \ \ 0\ \ \ 1\ \ \ldots 1\ 0\ \ \ 0 \\
Q_2 \quad 0\ \ 1\ \ \ldots 0\ \ 1\ \ 0\ \ \ 0\ \ 0\ \ \ 0\ \ \ 0\ 1\ 0\ \ \ 1\ 0\ 0 \\
\quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots
\end{array}
$$

The number of columns in the matrix will be $\Pi + S_1 + N_2$ where N_2 is the number of "type 2" factorizations and the number of columns is $N_1 + N_2$ where N_1 is the number of "type 1" factorizations. The matrix will again approach squareness and know that a row dependency will occur when the number of columns is equal to the number

of rows. Such a dependency represents a product of Q's which is a square and for which $\Pi(X-A)(X-B)$ is also a square.

Again we let P be the largest prime in the factor base; $\Pi = P/\log P$ is the number of primes in the factor base; and β is the parameter chosen so that

$$\text{(S4)} \qquad P = \left(\frac{S_1 + S_2}{2}\right)\sqrt{D}^{\,\beta}$$

For convenience, let $\bar{Q} = \left(\frac{S_1 + S_2}{2}\right)\sqrt{D}$.

If we fix B, the values of Q are of the form

$$\text{(S5)} \qquad Q = (K^2 + BK - D) + A(K + B)$$

and if p divides such a Q, it divides all Q of the form

$$Q = (K^2 + BK - D) + (A + np)(K + B),\ n \in Z$$

and only such Q. This enables us to factor all the S_1S_2 Q's by sieving on S_2 intervals of size S_1.

3.2. Running Analysis

To sieve on an interval, one must determine where to begin sieving for each prime p. This involves solving a congruence obtained by setting (S5) congruent to zero modulo p. This requires computing $(K + B)^{-1}$ modulo p which takes $\log p$ operations. Altogether, this requires

$$\text{(S6)} \qquad S_v = S_2 \sum_{p \le \Pi} \log p \le S_2\bar{Q}^{\beta} = S_2 P.$$

All the other timing computations are completely analogous to the continued fraction method. If x is the number of factored Q's required, it must satify

(S7)
$$1 = \frac{x}{S_1 + \Pi + x\,(1 - \frac{r(\alpha)}{r(\alpha,2\alpha)})}$$

or

(S8)
$$NF = x - \frac{(S_1+\Pi)\; r(\alpha,2\alpha)}{r(\alpha)} .$$

Then NQ, the number of Q's needed, will be

(S9)
$$NQ + NF/r(\alpha,2\alpha) = \frac{S_1 + \Pi}{r(\alpha)} .$$

For each prime p which divides a Q, we must perform two divisions; one to reduce the Q and one to determine that a higher power of p does not divide Q. Since the average number of primes dividing Q is $\log \log Q$, the total number of divisions can be estimated by

(S10) $ND = (2 \log \log \bar{Q})\; NQ$.

Of course care must be taken that NQ is not much larger than $S_1 S_2$ so there will be enough Q's to factor and for large numbers, S_2 must be chosen to be much less than S_1 so that S_v will be of the same order of magnitude as NQ.

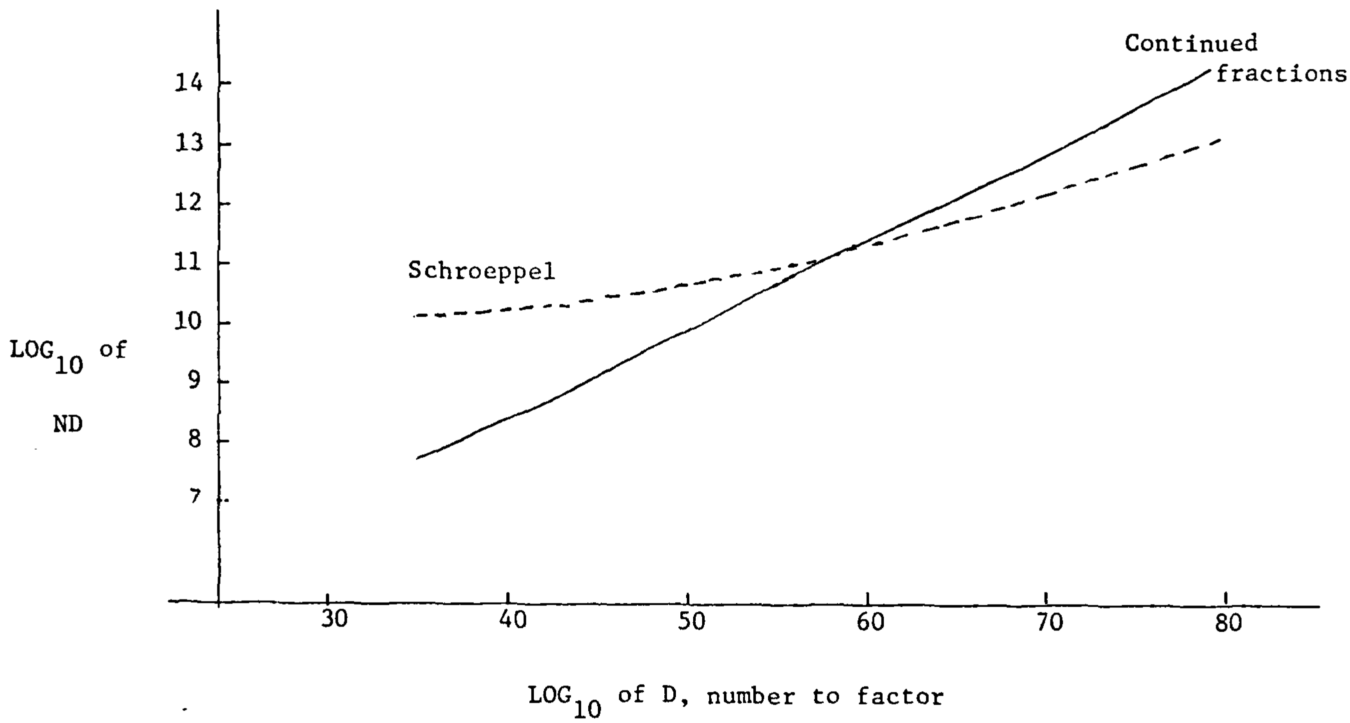
Table S1 shows some projections for factoring large numbers using this method. α was taken to be 1/6 throughout, and S_1 and S_2 were chosen to make NQ and $S_1 x S_2$ roughly equal and ND roughly equal to S_v. The total run time would be the sum of ND and S_v.

TABLE S1

D	S_1	S_2	P	Π	NF	NQ	$S_1 x S_2$	ND	SV
10^{40}	50,000	50,000	15,800	1,800	2,880,000	2.6 E 9	2.5 E 9	2.14E10	7.9 E 8
10^{50}	60,000	60,000	111,000	10,000	3,900,000	3.5 E 9	3.6 E 9	3.04E10	6.7 E 9
10^{60}	110,000	80,000	820,000	64,500	9,700,000	8.8 E 9	8.8 E 9	7.82E10	6.54E10
10^{70}	800,000	80,000	7,200,000	980,000	7.1E7	6.54E10	6.4 E10	5.9 E11	5.8 E11
10^{78}	3,000,000	80,000	28,400,000	1,750,000	2.6E8	2.42E11	2.40E11	2.24E12	2.27E12

In figure S1, we chart the running times of continued fraction versus Schroeppel. The solid line is continued fractions and the dotted line is Schroeppel's method.

FIG. S1



Although Schroeppel seems to be much faster for very large numbers, it requires a large amount of storage. For a 78 digit number, for example, 1,750,000 primes must be generated and sieving must be done on an interval of 3,000,000 bits. The method should be explored further, however.

IV. Anticipated Publications Related to the Research

1. "On Computing Unitary Aliquot Sequences", with R.K. Guy, Procedings of the tenth Manitoba Conference on Numerical Mathematics, 1979.

2. "An Analysis of a Simple Prime Proving Algorithm", submitted for publication.

3. "A Report on the Factorization of 2797 Numbers Using the Continued Fraction Algorithm", in preparation. (Draft included in the report).

4. "An Analysis of Pollard's Monte Carlo Factoring Method", in preparation.

5. "A Comparison of Two Factorization Methods", with S. Wagstaff Jr., to appear in Journal of Algorithms.

V. Personnel Associated With the Research Effort

P.T. Bateman, Head, Mathematics Department, The University of Illinois

Daniel Slotnick, Professor, Computer Science Department, The University of Illinois

S. Wagstaff, Assistant Professor, Mathematics Department, The University of Illinois

J. Godwin, Professor, Department of Computer Science and Statistics, Royal Holloway College, Egham, England

S.F. Reddaway, Research and Advanced Development Center, Stevenage, England

G. Lewis, Institute for Advanced Computation, Sunnyvale, California

VI. Interactions

The following is a chronological list of conferences attended and seminars given during the performance period of the grant.

"On Computing Unitary Aliguot Sequences", with R.K. Guy. The tenth Manitola Conference on Numerical Mathematics.

"Unitary Aliquot Sequences", talk given to the University of Illinois Number Theory Seminar, October 1979.

"An Analysis of the Continued Fraction Algorithm", talk given to the University of Illinois Combinatorics Computing Seminar, December, 1979.

"Factoring with Continued Fractions", talk given to the West Coast Number Theory Conference, December, 1979.

"Unitary Aliquot Sequences", talk given to the Royal Holloway College Maths Department, April, 1980.

"A New Method of Factoring", talk given to the Cardiff University Computer Science Department, May, 1980, Cardiff, Wales.

DA

FILM

7 –